ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Организация и проектирование СнК

Лабораторная работа №1

Вариант 11

Тестирование команды ORI

Выполнила:

студентка гр.Р41002

Стоколяс Ирина Андреевна

СПб, 2019

1. **Исходный код**

.data

init\_iregs: # сохранение данных для инициализации регистров в памяти.

.byte 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30

#используется директива byte т.к. размер данных не превышает 255

refs: # сохранение данных для проверки в памяти

.word 1, 2, 7, 12, 21, 38, 71, 136, 265, 522, 1035, 13, 15, 14, 15, 16, 49, 82, 147, 276, 533, 1046, 23, 26, 29, 26, 27, 60, 93, 158

save\_x2: # выделение в памяти места для сохранения значения регистра x2

.word 0

.text

main:

la x1, init\_iregs # загрузка адреса в x1 <-init\_regs

# команды загрузки значений из паямяти

lb x2, 0(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(0 \* 8 бит) => x2 = 1

lb x3, 1(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(1 \* 8 бит) => x3 = 2

lb x4, 2(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(2 \* 8 бит) => x4 = 3

lb x5, 3(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(3 \* 8 бит) => x5 = 4

lb x6, 4(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(4 \* 8 бит) => x6 = 5

lb x7, 5(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(5 \* 8 бит) => x7 = 6

lb x8, 6(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(6 \* 8 бит) => x8 = 7

lb x9, 7(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(7 \* 8 бит) => x9 = 8

lb x10, 8(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(8 \* 8 бит) => x10 = 9

lb x11, 9(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(9 \* 8 бит) => x11 = 10

lb x12, 10(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(10 \* 8 бит) => x12 = 11

lb x13, 11(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(11 \* 8 бит) => x13 = 12

lb x14, 12(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(12 \* 8 бит) => x14 = 13

lb x15, 13(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(13 \* 8 бит) => x15 = 14

lb x16, 14(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(14 \* 8 бит) => x16 = 15

lb x17, 15(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(15 \* 8 бит) => x17 = 16

lb x18, 16(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(16 \* 8 бит) => x18 = 17

lb x19, 17(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(17 \* 8 бит) => x19 = 18

lb x20, 18(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(18 \* 8 бит) => x20 = 19

lb x21, 19(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(19 \* 8 бит) => x21 = 20

lb x22, 20(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(20 \* 8 бит) => x22 = 21

lb x23, 21(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(21 \* 8 бит) => x23 = 22

lb x24, 22(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(22 \* 8 бит) => x24 = 23

lb x25, 23(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(23 \* 8 бит) => x25 = 24

lb x26, 24(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(24 \* 8 бит) => x26 = 25

lb x27, 25(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(25 \* 8 бит) => x27 = 26

lb x28, 26(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(26 \* 8 бит) => x28 = 27

lb x29, 27(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(27 \* 8 бит) => x29 = 28

lb x30, 28(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(28 \* 8 бит) => x30 = 29

lb x31, 29(x1) # загрузка значения из памяти по адресу init\_iregs + смешение(29 \* 8 бит) => x31 = 30

# выполнение команды ORI, т.к. I-immediate[11:0] в пределах [-2048; 2047](обозначено в компеляторе)

ori x2, x2, 0x001 # x2 = x2 ori 1 => x2 = 1

ori x3, x3, 0x002 # x3 = x3 ori 2 => x2 = 2

ori x4, x4, 0x004 # x4 = x4 ori 4 => x2 = 7

ori x5, x5, 0x008 # x5 = x5 ori 8 => x2 = 12

ori x6, x6, 0x010 # x6 = x6 ori 16 => x2 = 21

ori x7, x7, 0x020 # x7 = x7 ori 32 => x2 = 38

ori x8, x8, 0x040 # x8 = x8 ori 62 => x2 = 71

ori x9, x9, 0x080 # x9 = x9 ori 128 => x2 = 136

ori x10, x10, 0x100 # x10 = x10 ori 256 => x2 = 265

ori x11, x11, 0x200 # x11 = x11 ori 512 => x2 = 522

ori x12, x12, 0x400 # x12 = x12 ori 1024 => x2 = 1035

ori x13, x13, 0x001 # x13 = x13 ori 1 => x2 = 13

ori x14, x14, 0x002 # x14 = x14 ori 2 => x2 = 15

ori x15, x15, 0x004 # x15 = x15 ori 4 => x2 = 14

ori x16, x16, 0x008 # x16 = x16 ori 8 => x2 = 15

ori x17, x17, 0x010 # x17 = x17 ori 16 => x2 = 16

ori x18, x18, 0x020 # x18 = x18 ori 32 => x2 = 49

ori x19, x19, 0x040 # x19 = x19 ori 62 => x2 = 82

ori x20, x20, 0x080 # x20 = x20 ori 128 => x2 = 147

ori x21, x21, 0x100 # x21 = x21 ori 256 => x2 = 276

ori x22, x22, 0x200 # x22 = x22 ori 512 => x2 = 533

ori x23, x23, 0x400 # x23 = x23 ori 1024 => x2 = 1046

ori x24, x24, 0x001 # x24 = x24 ori 1 => x2 = 23

ori x25, x25, 0x002 # x25 = x25 ori 2 => x2 = 26

ori x26, x26, 0x004 # x26 = x26 ori 4 => x2 = 29

ori x27, x27, 0x008 # x27 = x27 ori 8 => x2 = 26

ori x28, x28, 0x010 # x28 = x28 ori 16 => x2 = 27

ori x29, x29, 0x020 # x29 = x29 ori 32 => x2 = 60

ori x30, x30, 0x040 # x30 = x30 ori 64 => x2 = 93

ori x31, x31, 0x080 # x31 = x31 ori 128 => x2 = 158

la x1, save\_x2 # запись адреса x1 <- save\_x2

sw x2, 0(x1) # запись значения по адресу

la x1, refs # запись адреса x1 <- refs

# загрузка значений для тестирования и тестирование

lw x2, 4(x1) # загрузка значения из памяти по адресу refs + смешение(4 \* 8 бит) => x2 = 2

bne x3, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x3

lw x2, 8(x1) # загрузка значения из памяти по адресу refs + смешение(8 \* 8 бит) => x2 = 7

bne x4, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x4

lw x2, 12(x1) # загрузка значения из памяти по адресу refs + смешение(12 \* 8 бит) => x2 = 12

bne x5, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x5

lw x2, 16(x1) # загрузка значения из памяти по адресу refs + смешение(16 \* 8 бит) => x2 = 21

bne x6, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x6

lw x2, 20(x1) # загрузка значения из памяти по адресу refs + смешение(20 \* 8 бит) => x2 = 38

bne x7, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x7

lw x2, 24(x1) # загрузка значения из памяти по адресу refs + смешение(24 \* 8 бит) => x2 = 71

bne x8, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x8

lw x2, 28(x1) # загрузка значения из памяти по адресу refs + смешение(28 \* 8 бит) => x2 = 136

bne x9, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x9

lw x2, 32(x1) # загрузка значения из памяти по адресу refs + смешение(32 \* 8 бит) => x2 = 265

bne x10, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x10

lw x2, 36(x1) # загрузка значения из памяти по адресу refs + смешение(36 \* 8 бит) => x2 = 522

bne x11, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x11

lw x2, 40(x1) # загрузка значения из памяти по адресу refs + смешение(40 \* 8 бит) => x2 = 1035

bne x12, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x12

lw x2, 44(x1) # загрузка значения из памяти по адресу refs + смешение(44 \* 8 бит) => x2 = 13

bne x13, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x13

lw x2, 48(x1) # загрузка значения из памяти по адресу refs + смешение(48 \* 8 бит) => x2 = 15

bne x14, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x14

lw x2, 52(x1) # загрузка значения из памяти по адресу refs + смешение(52 \* 8 бит) => x2 = 14

bne x15, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x15

lw x2, 56(x1) # загрузка значения из памяти по адресу refs + смешение(56 \* 8 бит) => x2 = 15

bne x16, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x16

lw x2, 60(x1) # загрузка значения из памяти по адресу refs + смешение(60 \* 8 бит) => x2 = 16

bne x17, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x17

lw x2, 64(x1) # загрузка значения из памяти по адресу refs + смешение(64 \* 8 бит) => x2 = 49

bne x18, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x18

lw x2, 68(x1) # загрузка значения из памяти по адресу refs + смешение(68 \* 8 бит) => x2 = 82

bne x19, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x19

lw x2, 72(x1) # загрузка значения из памяти по адресу refs + смешение(72 \* 8 бит) => x2 = 147

bne x20, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x20

lw x2, 76(x1) # загрузка значения из памяти по адресу refs + смешение(76 \* 8 бит) => x2 = 276

bne x21, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x21

lw x2, 80(x1) # загрузка значения из памяти по адресу refs + смешение(80 \* 8 бит) => x2 = 533

bne x22, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x22

lw x2, 84(x1) # загрузка значения из памяти по адресу refs + смешение(84 \* 8 бит) => x2 = 1046

bne x23, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x23

lw x2, 88(x1) # загрузка значения из памяти по адресу refs + смешение(88 \* 8 бит) => x2 = 23

bne x24, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x24

lw x2, 92(x1) # загрузка значения из памяти по адресу refs + смешение(92 \* 8 бит) => x2 = 26

bne x25, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x25

lw x2, 96(x1) # загрузка значения из памяти по адресу refs + смешение(96 \* 8 бит) => x2 = 29

bne x26, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x26

lw x2, 100(x1) # загрузка значения из памяти по адресу refs + смешение(100 \* 8 бит) => x2 = 26

bne x27, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x27

lw x2, 104(x1) # загрузка значения из памяти по адресу refs + смешение(104 \* 8 бит) => x2 = 27

bne x28, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x28

lw x2, 108(x1) # загрузка значения из памяти по адресу refs + смешение(108 \* 8 бит) => x2 = 60

bne x29, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x29

lw x2, 112(x1) # загрузка значения из памяти по адресу refs + смешение(112 \* 8 бит) => x2 = 93

bne x30, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x30

lw x2, 116(x1) # загрузка значения из памяти по адресу refs + смешение(116 \* 8 бит) => x2 = 158

bne x31, x2, test\_failed # условный переход по адресу метки test\_failed если x2 != x31

# проверка значений х2

lw x3, 0(x1) # загрузка значения из памяти по адресу refs + смешение(0 \* 8 бит) => x3 = 1

la x1, save\_x2 # запись адреса x1 <- save\_x2

lw x2, 0(x1) # загрузка значения из памяти по адресу refs + смешение(0 \* 8 бит) => x2 = 1

bne x3, x2, test\_failed # условный переход по адресу метки test\_failed если x3 != x2

test\_successful:

li a0, 1 # системный вызов: print\_int - печать числа

la a1, 0xB000 # загрузка в a1 значения для сис.вызова 0xB000 = 45056

ecall # системный вызов

j exit # безусловный переход к меке exit

test\_failed:

li a0, 4 # системный вызов: print\_character - печать символа

li a1, 'N' # загрузка в a1 значения для сис.вызова символ N

ecall # системный вызов

exit:

li a0, 10 # системный вызов: exit - остановка симулятора

ecall # системный вызов

1. **Ссылка на github**

<https://github.com/wildistok/RISC-V-Assembler>

1. **Описание стратегии тестирования**

Для тестирования вручную было выполнено побитовое или для значений в регистрах x2-x31 и immediate части команды.

Описание на примере в hex представлении данных для данных не более 2^12/2, т.к. immediate значения не могут быть больше и соответственно результат побитового ИЛИ также не будет больше 2^12/2(с учетом ограничений симулятора):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| reg | 0x001 | 0x002 | 0x003 | 0x004 |
| immediate | 0x001 | 0x002 | 0x004 | 0x008 |
| result | 0x001 | 0x002 | 0x007 | 0x00C |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| reg | 0x005 | 0x006 | 0x007 | 0x008 |
| immediate | 0x010 | 0x020 | 0x040 | 0x080 |
| result | 0x015 | 0x026 | 0x047 | 0x088 |

|  |  |  |  |
| --- | --- | --- | --- |
| reg | 0x009 | 0x00A | 0x00B |
| immediate | 0x100 | 0x200 | 0x400 |
| result | 0x109 | 0x20A | 0x40B |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| reg | 0x00C | 0x00D | 0x00E | 0x00F |
| immediate | 0x001 | 0x002 | 0x004 | 0x008 |
| result | 0x00D | 0x00F | 0x00E | 0x00F |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| reg | 0x010 | 0x011 | 0x012 | 0x013 |
| immediate | 0x010 | 0x020 | 0x040 | 0x080 |
| result | 0x010 | 0x031 | 0x052 | 0x093 |

|  |  |  |  |
| --- | --- | --- | --- |
| reg | 0x014 | 0x015 | 0x016 |
| immediate | 0x100 | 0x200 | 0x400 |
| result | 0x114 | 0x215 | 0x416 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| reg | 0x017 | 0x018 | 0x019 | 0x01A |
| immediate | 0x001 | 0x002 | 0x004 | 0x008 |
| result | 0x017 | 0x01A | 0x01D | 0x01A |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| reg | 0x01B | 0x01C | 0x01D | 0x01E |
| immediate | 0x010 | 0x020 | 0x040 | 0x080 |
| result | 0x01B | 0x03C | 0x05D | 0x09E |

Полученные значения переводим в десятичный формат и используем в программе в качестве ожидаемых результатов для соответствующих начальных значений.

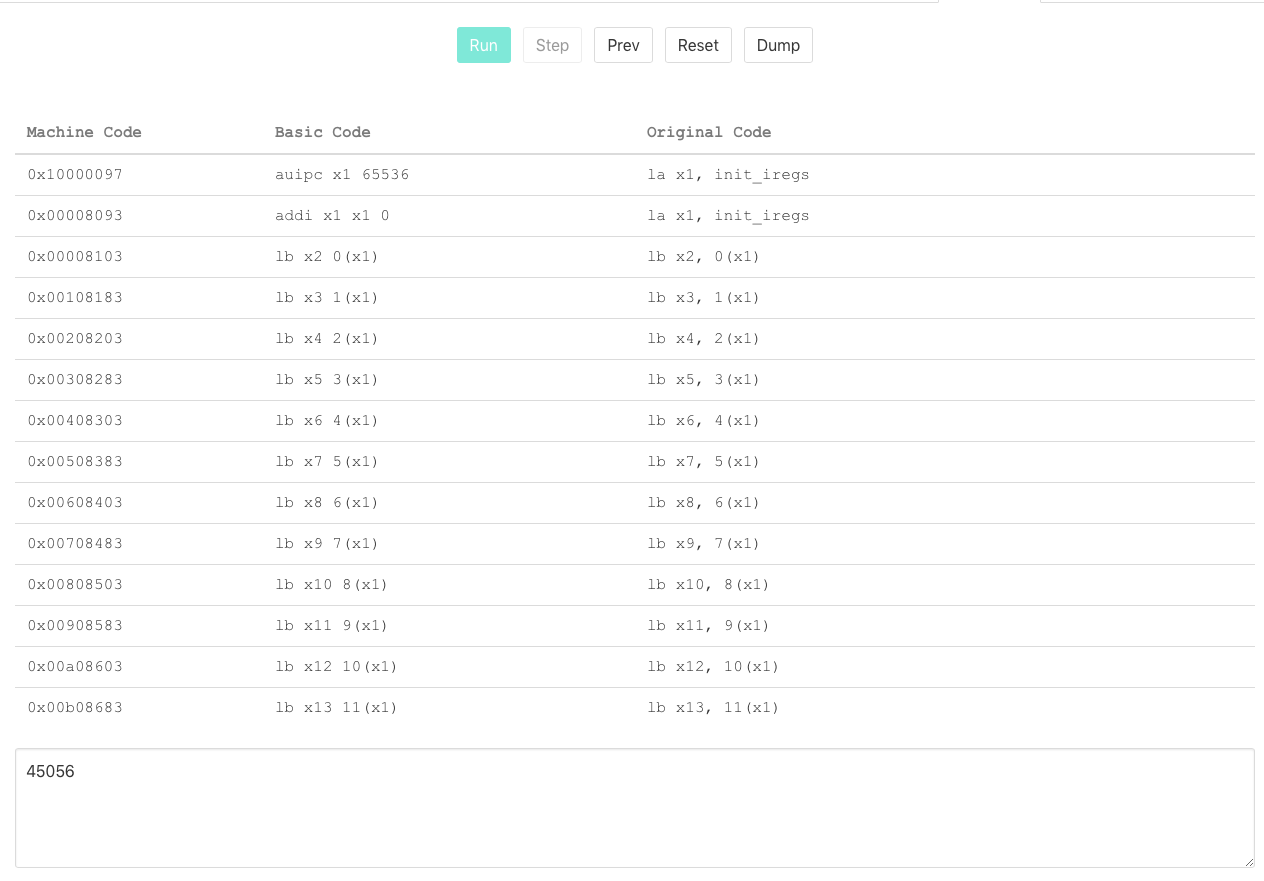
Проверка осуществляется с помощью команд условного перехода при сравнении полученных и ожидаемых значений.

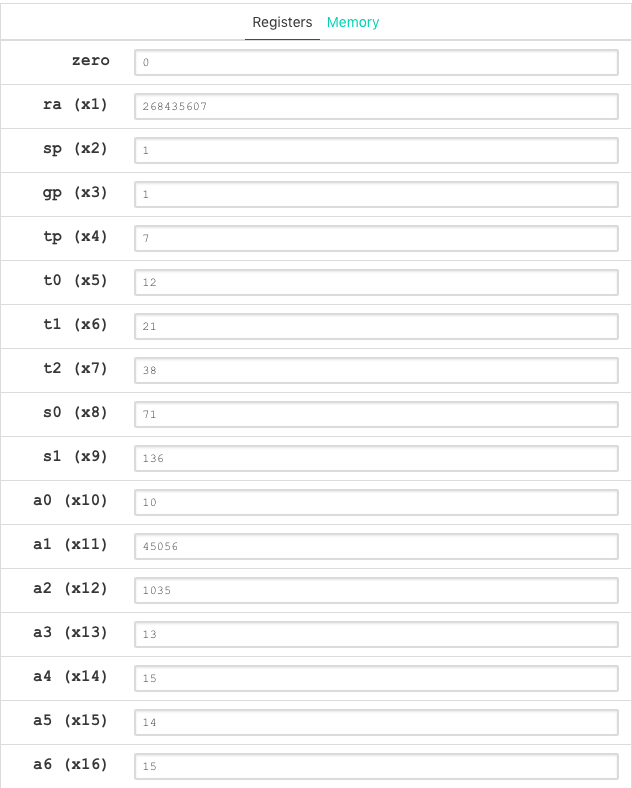
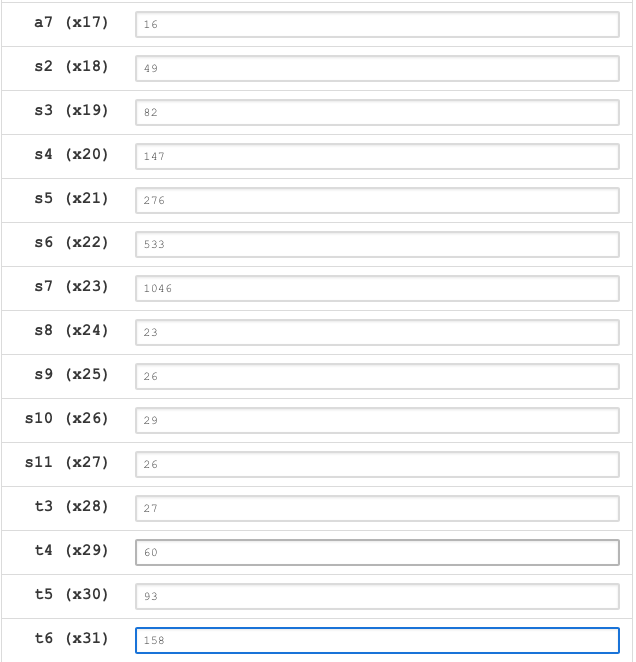
Для хранения ожидаемых значений используем регистр x2. Для проверки значения x2 сначала сохраняем его в память а потом востанавливаем и осуществляем проверку (т.к. количество регистров недостаточно для хранения ожидаемых результатов).

В случае успешного выполнения всех проверок с помощью системного вызова выводим в консоль симулятора десятичное значение числа 0xB000. В случае, если хотя бы одна проверка не была пройдена выводим в консоль символ “N”.

1. **Результаты выполнения**

При успешном прохождении всех тестовых проверок:

* Вывод в консоль(также на скриншоте представлена часть выполняемой программы)
* Содержимое регистров



* Содержимое памяти

Область .data



Область .text

